International Journal of Reliability and Applications
Vol. 3, No. 3, pp. 133-145, 2002

Performance Analysis for Flow Networks by Most Probable States

Hee Kyoung Lee, Dong Ho Park and Seung Min Lee *
Department of Statistics, Hallym University,
Chunchon 200-702, Korea

Abstract. The traditional methods of evaluating the performance of a network by enumerating all possible states may quickly become computationally prohibitive, since the number of states grows exponentially as the number of components increases. In such cases, enumerating only the most probable states would provide a good approximation. In this paper, we propose a method which efficiently generates upper and lower bounds for coherent performance measures utilizing the most probable states. Compared with Yang and Kubat's method, our procedure significantly reduces the complexity and memory requirement per iteration for computing the bounds and thereby, achieves the given degree of accuracy or the coverage within a shorter time.

Key Words : *reliability, performance, link capacity, maximum flow, most probable states*

1. INTRODUCTION

A network is represented by a probabilistic graph $G(V, E)$, which consists of a set V of nodes and a set E of links. Each link of the network may have different flow capacity. Each link of the network either operates or fails with known probability. The network is required to transmit a specified amount of flow from the source node to the terminal node, and the maximum amount of network flow which can be transmitted from the source node to the terminal node is called the maximum flow of the network. The number of possible states of a flow network with n links is 2^n and, due to the large size of state space even for a network of moderate size, the evaluation of network performance by enumerating all possible network states would seem impractical for a network with a large number of links. Li and Silvester(1984) suggest a method which considers only m most probable states which guarantee a certain coverage of the state space. Since these states of high probability account

*Corresponding author. *E-mail address:* smlee1@hallym.ac.kr

for a large fraction of the total probability, this approach is considered effective in practice. However, Li and Silvester's algorithm lacks flexibility in generating most probable states for computing the bounds, since if the number of most probable states to be generated is changed, the whole process must start again from its first step. The method of Li and Silvester is improved by Lam and Li(1986) , Shier(1988), and Gomes and Craveririnha(1998) in efficiency and flexibility. Lam and Li's algorithm generates the candidate states of most probable states, stores them in heap and then obtain the current most probable states one by one in decreasing order of state probability. Shier's method suggests to enumerate the most probable states one by one from a smaller group of candidate states. The method of Gomes and Craveririnha enumerates the most probable states from the candidate states stored in heap using successive order.

Notations

$p_j(\frac{1}{2} \leq p_j \leq 1)$	probability that link j operates $(j = 1, 2, \cdots, n)$
$q_j = 1 - p_j$	probability that link j fails $(j = 1, 2, \cdots, n)$
$S_i = \{i_1, i_2, \cdots, i_k\}$	set of failed links in ith MPS $(i = 1, \cdots, 2^n)$
	where $1 \leq k \leq n$, $1 \leq i_k \leq n (i_1 < i_2 < \cdots < i_k)$
$S_i^P = \{i_1, i_2, \cdots, i_{k-1}\}$	set S_i with element i_k deleted
$P(S_i)$	probability of S_i
S_i^l	set of reduced states of S_i
$P_L(S_i)$	probability of S_i^l
$R(S_i)(R(S_i^P))$	flow in S_i(flow in S_i^P)
$U(i)$	upper bound at ith iteration
$L(i)$	lower bound at ith iteration
$\bar{R}$	performance measurement of network(performability)
	(example : expected maximum flow)
$\alpha = R(S_1)$	maximal value of $R(S_i)$ $(i = 1, \cdots, 2^n)$
$\beta = R(S_{2^n})$	minimal value of $R(S_i)$ $(i = 1, \cdots, 2^n)$

Given a state S, the probability of S, $P(S)$, is given by

$$P(S) = \prod_{i=1}^{n} p_i^{x_i} \cdot q_i^{1-x_i}$$

and the states S_i, $i = 1, 2, \cdots, 2^n$, are arranged in decreasing order as

$$P(S_1) \geq P(S_2) \geq \cdots \geq P(S_{2^n}).$$

Clearly, the most probable state in the whole state space is $S_1 = \phi$ with $P(S_1) = \prod_{i=1}^{n} p_i$, and the next most probable state is $S_2 = \{1\}$ with $P(S_2) = q_1 \prod_{i=2}^{n} p_i$. Simi-

larly, the least most probable state is $S_{2^n} = \{1, 2, \cdots, n\}$ with $P(S_{2^n}) = \prod_{i=1}^{n} q_i$. A performance measure, R say, satisfies a coherence property if

$$R(S_i) \geq R(S_j) \text{ for } S_i \geq S_j$$

and we define the performability of a network as

$$\bar{R} = \sum_{i=1}^{2^n} R(S_i) \cdot P(S_i).$$

For large n, it would be difficult to evaluate the exact value of $\bar{R}$ in reasonable time. Li and Silvester(1984) enumerate S_1, S_2, $\cdots$, S_m in iterations and, at each iteration, an upper bound $U(m)$ and a lower bound $L(m)$ are derived as follows;

$$U(m) = \sum_{i=1}^{m} R(S_i) \cdot P(S_i) + [1 - \sum_{i=1}^{m} P(S_i)] \times \alpha$$

and

$$L(m) = \sum_{i=1}^{m} R(S_i) \cdot P(S_i) + [1 - \sum_{i=1}^{m} P(S_i)] \times \beta,$$

where m is the number of most probable states generated and satisfies

$$\sum_{i=1}^{m} P(S_i) \geq \Phi(coverage) \text{ or } \Delta_{\bar{R}} \simeq \frac{U-L}{L} \leq \varepsilon.$$

With a performance measure having coherence property, Yang and Kubat(1990) provide superior upper and lower bounds. They generate the most probable states through tree search, and provide converging upper and lower bounds for performability at each iteration.

In this paper, we provide a new algorithm for computing upper and lower bounds, which siginificantly reduces the complexity and memory requirement per iteration than the existing methods. In Section 2, we describe Yang and Kubat's algorithm. In Section 3, we present our algorithm and compares our method with that of Yang and Kubat. Some numerical examples are discussed in Section 4.

2. YANG AND KUBAT'S ALGORITHM

Considering a network having n links, we make the following assumptions.

1. Each network link can be in one of two states, operational or failed
2. The links fail independently of each other

Yang and Kubat(1990) calculate the most probable states through tree search. More

specifically, given n and p_i for $i = 1, 2, \cdots, n$, they define a special binary tree G with height n, with each state having exactly two sons and each vertex at level n is a leaf.(Clearly, this tree has 2^n leaves.) The tree structure for the bridge network of Figure 3 is shown below in Figure 1.

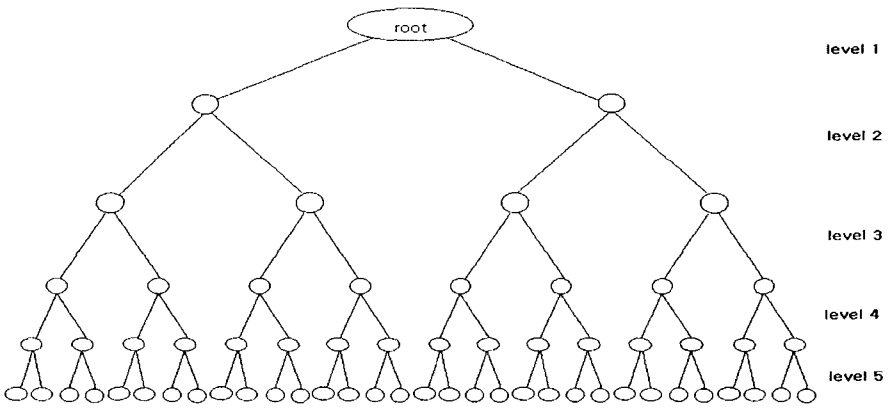

Figure 1. Tree structure of bridge network

The weight of vertex u at level l, denoted as $w(u)$, is defined to be

$$w(u) = \prod_{i=1}^{l} p_i^{x_i} \cdot q_i^{1-x_i}.$$

Clearly, $w(u)$ corresponds to the weight sum of all the leaves in G_u. Also, we have that $w(u) = P(S)$. Thus, the problem of enumerating the most probable states is equivalent to identifying the addresses of the heaviest leaves in G. In Yang and Kubat's algorithm, the address of the mth heaviest leaf of G is identified at the mth iteration of the algorithm. For a vertex u of G at level l, the heaviest leaf in G_u has weight

$$w(u) \cdot \prod_{i=l+1}^{n} p_i.$$

Yang and Kubat's algorithm identifies the currently most probable states at each iteration and provides converging upper and lower bounds for the expected maximum flow after each iteration. Suppose that we are at the beginning of the mth iteration of the algorithm and each internal vertex u of G is associated with four variables.

- $W_1(u) = w(u) \cdot p_{l+1} \cdot \prod_{i=l+2}^{n} p_i$
- $U_1(u) = w(u) \cdot p_{l+1} \cdot \alpha$

- $W_0(u) = w(u) \cdot q_{l+1} \cdot \prod_{i=l+2}^{n} p_i$
- $U_0(u) = w(u) \cdot q_{l+1} \cdot \alpha$

$W_1(u)(W_0(u))$ indicates the weight of the heaviest remaining leaf in the left(right) subtree of u in G. $U_1(u)(U_0(u))$ indicates upper bound of the left(right) son. If vertex u has not yet been created in G, we create it. Suppose that vertex u has already been created in G. Then we visit the left son of u if $W_1(u) \geq W_0(u)$, and the right son otherwise. When finally a leaf, say v, is reached, we can be sure that v corresponds to the heaviest remaining leaf of G. Calculate $R(v)$ in leaf v and to begin update. We only need to update $U(v)$ by $w(v) \cdot R(v)$.
After each iteration of the algorithm, we output $U(m)$ and $L(m)$ as follows.

$$U(m) = U_1(root) + U_0(root)$$

and

$$L(m) = \sum_{i=1}^{m} R(S_i)w(S_i) + [1 - \sum_{i=1}^{m} w(S_i)] \cdot \beta.$$

3. ALGORITHM

Let G be a *binary tree* and consider two nodes u and v in a tree G. If node u is the *parent* of node v, then we say that v is a *child* of u. Two nodes that are children of the same parent are *siblings*. The node u is an *ancestor* of the node v if there exists a path in G from u to v. Similarly, node v is a *descendant* of the node u if there exists a path in G from u to v. A node that has no sons is called a *leaf*. A node u and all its descendants, denoted by G_u, is called a *subtree* of G. A tree structure is used to compute the upper and lower bounds. The tree structure of bridge network of Figure 3 for bounds computation, is shown below in Figure 2.

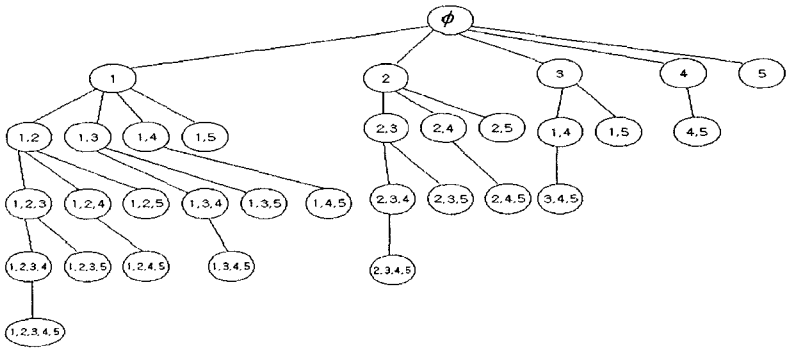

Figure 2. Tree structure of bridge network of Figure 3

Given the most probable states, our algorithm calculates the upper bound at each iteration independently by the flow of parent nodes. Our algorithm will be more effective in the memory size and the executing time. To compare the efficiency of our method, we use Yang and Kubat's algorithm SEARCH_MPS(). SEARCH_FLOW() is an algorithm that calculates the flow(R) for the given S_i.

SEARCH_MPS(tree, l, w)

if $l == n$ then return tree

if tree has not been visited before then create and let

$$tree\_W_1[l] = w \cdot p_{l+1} \cdot P[l+1]$$
$$tree\_W_0[l] = w \cdot (1 - p_{l+1}) \cdot P[l+1]$$
$$tree\_U_1[l] = w \cdot p_{l+1} \cdot \alpha$$
$$tree\_U_0[l] = w \cdot (1 - p_{l+1}) \cdot \alpha$$

else choose k such that $tree\_W_k[l] = max[tree\_W_1[l], tree\_W_0[l]]$

$$tree' \leftarrow tree\_W_k[l]$$
$$y_{l+1} \leftarrow k$$

if $k == 0$ then $w = w \cdot (1 - p_{l+1})$, $P_L(S_i) = w$

else $w = w \cdot p_{l+1}$

Call SEARCH_MPS($tree'$, $l+1$, w)

Algorithm

Step 1. Input : ε, p_j, $j = 1, 2, \cdots, n$

Step 2. Initialize : $U(0) = \alpha$, $L(0) = 0$, $i = 0$, $\Delta_{\bar{R}} = 1$

Step 3. Create :

an n vector $(y_1, y_2, \cdots, y_n)$
for $l = n - 1$ to 0 do $P[l] = p_{l+1} \cdot P[l+1]$
a vertex root

$$root\_W_1[0] = P[0]$$
$$root\_W_0[0] = P[0]\frac{(1-p_1)}{p_1}$$
$$root\_U_1[0] = P[0] \cdot \alpha$$
$$root\_U_0[0] = (1 - p_1) \cdot \alpha$$

Step 4. Repeat :

Initialize : $P_L(S_i) = w = 1$, $R = null$
$i = i + 1$
Call SEARCH_MPS(root, 0, 1);
Call SEARCH_FLOW(S_i);
$R(S_i^P)$ calculation at process that Yang and Kubat's algorithm does update
if $R == null$ then $R = R(S_i)$ else $R(S_i^P) = R$
if $R(S_i) \neq R(S_i^P)$ then
$U(i) = U(i-1) + P_L(S_i) \times (R(S_i) - R(S_i^P))$
$L(i) = L(i-1) + w \times R(S_i)$
$\Delta_{\bar{R}} = \frac{U(i)-L(i)}{L(i)}$
if $\Delta_{\bar{R}} \geq \varepsilon$ then go to **Step 4** else Stop.

Lemma 1. Upper bound $U(i)$ can be expressed as follows.

$$U(i) = U(i-1) + P_L(S_i) \times (R(S_i) - R(S_i^P))$$

Proof. Suppose that the number of son nodes of node u is m, and mark each of them by $u_1, u_2, \cdots, u_m$. Total probability in node u is $P_L(u)$, and probability of state is $P(u)$, and flow is $R(u)$. The total probability of each son node is $P_L(u_1), \cdots, P_L(u_m)$, and flow is $R(u_1), \cdots, R(u_m)$. Therefore, being $R(u) \geq R(u_1), \cdots, R(u) \geq R(u_m)$ by coherence property, the total probability is $P_L(u) = P_L(u_1) + \cdots + P_L(u_m) + P(u)$. Then,

$$\begin{aligned}
U(i-1) &= RU + P_L(u) \cdot R(u) \\
&= RU + (P_L(u_1) + \cdots + P_L(u_m) + P(u))R(u) \\
&= RU + P_L(u_1)R(u) + (P_L(u_2) + \cdots + P_L(u_m) + P(u))R(u) \\
&\geq RU + P_L(u_1)R(u_1) + (P_L(u_2) + \cdots + P_L(u_m) + P(u))R(u) \\
U(i) &= RU + P_L(u_1)R(u_1) + (P_L(u_2) + \cdots + P_L(u_m) + P(u))R(u) \\
&= RU + P_L(u_1)R(u_1) + (P_L(u) - P_L(u_1))R(u) \\
&= RU + P_L(u_1)R(u_1) + P_L(u)R(u) - P_L(u_1)R(u) \\
&= U(i-1) + P_L(u_1)(R(u_1) - R(u))
\end{aligned}$$

4. NUMERICAL EXAMPLES

Example 1. We consider the bridge network in Figure 3, with the flow capacities $c_1 = 2$, $c_2 = 6$, $c_3 = 4$, $c_4 = 5$, $c_5 = 3$. Let $p_1 = 0.5$, $p_2 = 0.6$, $p_3 = 0.7$, $p_4 = 0.8$, $p_5 = 0.9$. The number of states is $2^5 = 32$, $\bar{R} = 3.4952$, $\varepsilon \geq 0.05$.

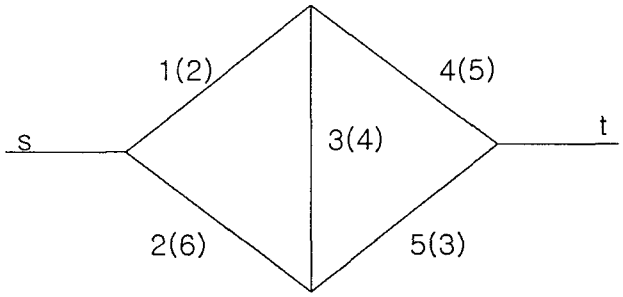

Figure 3. Bridge network

Figure 4 and Figure 6 show the process to find the most probable states. Figure 5 and Figure 7 show the update process.

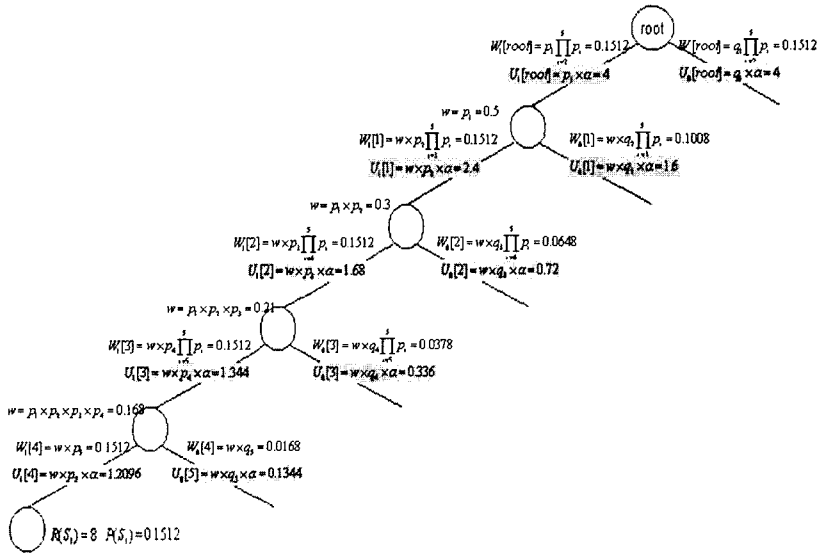

Figure 4. MPS at iteration 1

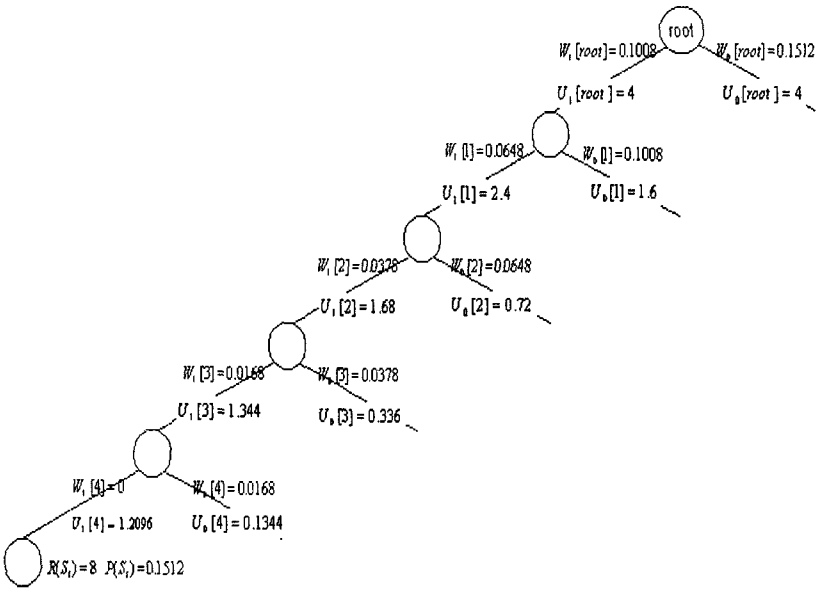

Figure 5. Update at iteration 1

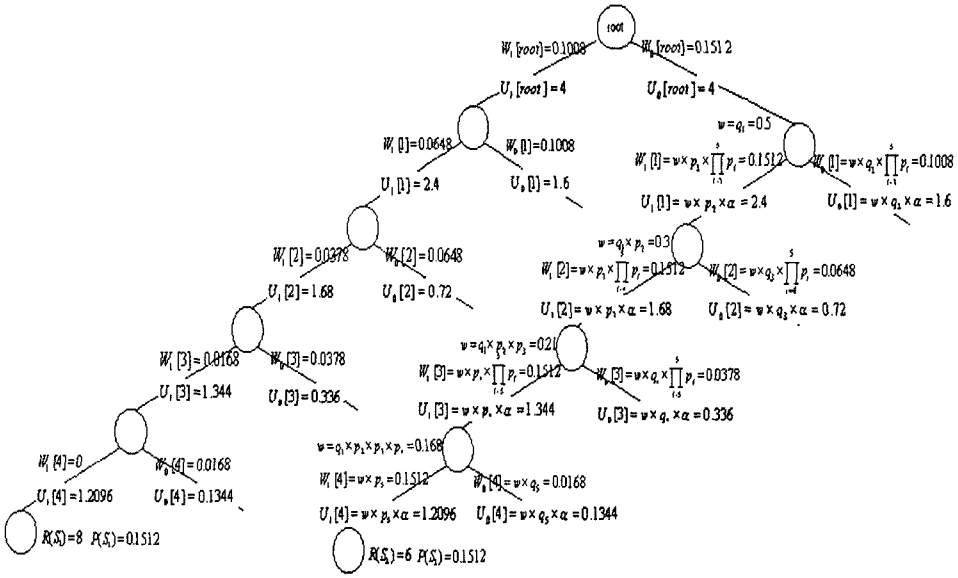

Figure 6. MPS at iteration 2

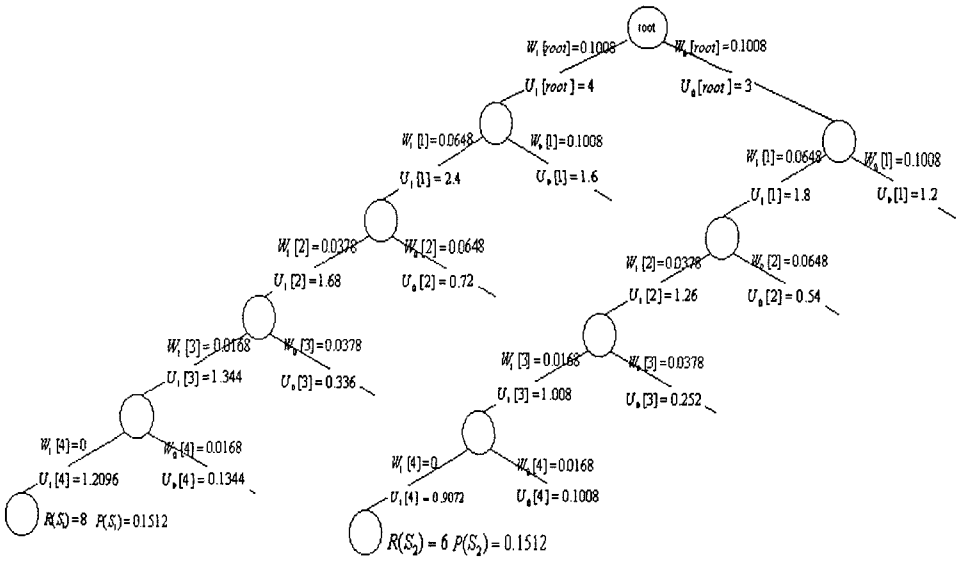

Figure 7. Update at iteration 2

As can be seen in Figure 8 and Table 1, our algorithm is more efficient in generating the upper and lower bounds at each iteration.

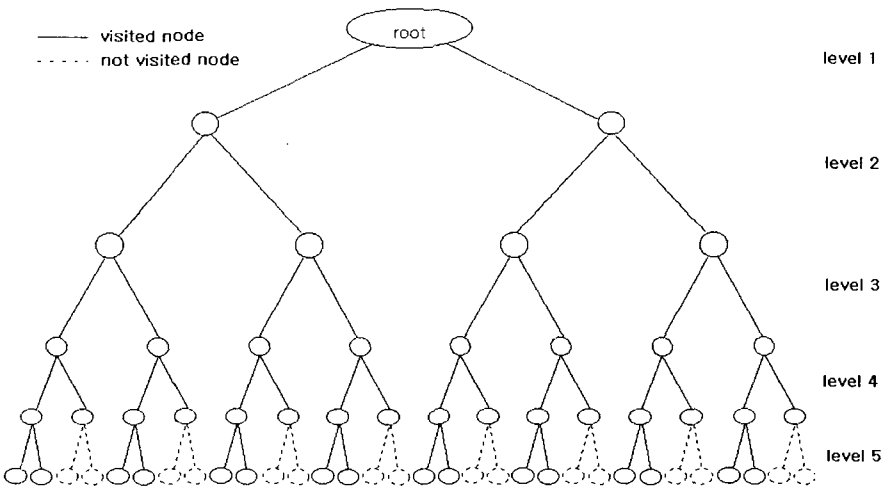

Figure 8. Yang and Kubat's method

Table 1. Our method

Iteration(i)	State(S_i)	Flow($R(S_i)$)	$P_L(S_i)$	Calculation
1	ϕ	8		8
2	1	6	0.5	$8 + (0.5) \times (6 - 8) = 7$
3	2	2	0.2	$7 + (0.2) \times (2 - 8) = 5.8$
4	1,2	0	0.2	$5.8 + (0.2) \times (0 - 6) = 4.6$
5	3	5	0.09	$4.6 + (0.09) \times (5 - 8) = 4.33$
6	1,3	3	0.09	$4.33 + (0.09) \times (3 - 6) = 4.06$
7	2,3	2	0.06	$4.06 + (0.06) \times (2 - 2) = 4.06$
8	1,2,3	0	0.06	$4.06 + (0.06) \times (0 - 0) = 4.06$

Table 2 campares our method with that of Yang and Kubat for generating the same number of most probable states in time and coverage.
(Computer : Pentium III-450 Dual CPU, RAM 128M, wowlinux7.0, gcc 2.96)

Table 2. Bridge network results

	Yang & Kubat	our method
number of used MPS	16	16
upper bound	3.604	3.604
lower bound	3.4488	3.4488
coverage	0.912	0.912
mean time(sec.)	0.000105	0.000093

Example 2. Consider a network given in Figure 9 with $p_1 = 0.8$, $p_2 = 0.85$, $p_3 = 0.9$, $p_4 = 0.95$, $p_5 = 0.99$, $p_6 = 0.999$, $p_7 = 0.999$, $p_8 = 0.999$, $p_9 = 0.999$. In this example, we consider the following performance measure.

$$R(S_i) = \begin{cases} 1, & \text{if the network is connected given that the network state is } S_i \\ 0, & \text{if not} \end{cases}$$

The number of whole state space is $2^9 = 514$ and $\bar{R} = 0.979828$. Tables 3 and 4 compare our method with Yang and Kubat's algorithm. It shows that our algorithm significantly reduces the memory size and execution time, so that it achieves the given degree of accuracy or the coverage within a shorter time(See Table 3.) or the performability $\bar{R}$(See Table 4.).

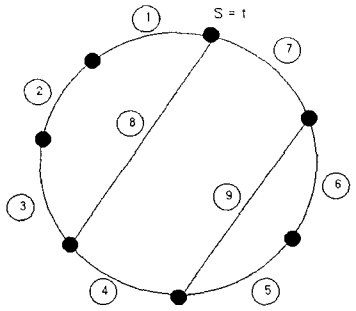

Figure 9. Network

Table 3. Comparison for the network of Figure 9

	$\varepsilon = 0.001$		$\varepsilon = 0.00001$	
	Yang & Kubat	new	Yang & Kubat	new
number of used MPS	33	33	88	102
upper bound	0.980220	0.980313	0.979831	0.979833
lower bound	0.979339	0.979339	0.979822	0.979823
coverage	0.998755	0.998755	0.999710	0.999986
mean time(sec.)	0.000307	0.000218	0.000587	0.000432

Table 4. Comparison results for $\bar{R}$=0.979828

	upper bound		*lower bound*	
	Yang & Kubat	new	Yang & Kubat	new
number of used MPS	111	146	119	119
coverage	0.999991	0.999998	0.999994	0.999994
mean time(sec.)	0.000705	0.000570	0.000718	0.000504

REFERENCES

Li, V.O.K. and Silvester, J.A. (1984), Performance analysis networks with unreliable components, *IEEE Transactions on Communications*, Vol. COM-32, pp 1105-1110.

Lam, Y.F. and Li, V.O.K. (1986), An improved algorithm for performance analysis of networks with unreliable components, *IEEE Transactions on Communications*, Vol. COM-34, pp 496-497.

Shier, D.R. (1988), A new algorithm for performance analysis of communication systems, *IEEE Transactions on Communications*, Vol. COM-36, pp 516-519.

Gomes, T.M.S. and Craveirinha, J.M.F. (1998), Algorithm for sequential generation of states in failure-prone communication network, *IEEE Proc Commun.*, Vol. 145, pp 73-79.

Yang, C.L. and Kubat, P. (1990), An algorithm for network reliability bounds, *Operations Research Society of America Journal on computing*, Vol. 2, No. 4, fall.